(12) UK Patent Application (19) GB (11) 2 271 653 (13) A

(43) Date of A Publication 20.04.1994

(21) Application No 9320511.0

(22) Date of Filing 05.10.1993

(30) Priority Data
(31) 07960598 (32) 13.10.1992 (33) US

(51) INT CL[5]
G06F 12/08

(52) UK CL (Edition M)
G4A AMC

(56) Documents Cited
GB 2037038 A GB 1449877 A EP 0077451 A2
US 4197580 A US 4075686 A

(58) Field of Search
UK CL (Edition L) G4A AMC
INT CL[5] G06F 12/08

(71) Applicant(s)
Hewlett-Packard Company

(Incorporated in USA - California)

3000 Hanover Street, Palo Alto, California 94304, United States of America

(72) Inventor(s)
Rau B Ramakrishna
Michael S Schlansker
Williams S Worley Jr

(74) Agent and/or Address for Service
Williams,Powell & Associates
34 Tavistock Street, LONDON, WC2E 7PB, United Kingdom

(54) Multi-level cache system

(57) A cache system 10 for buffering data transfers between a CPU 12 and main memory 13 has e.g. two levels of buffers 15, 16. Data may be transferred via the highest level buffer. At the second level, data is transferred between the highest level and the CPU if the data is present at the highest level. However, if the data is not present at the highest level, the data is transferred from the second level buffer without making copies of the data in the first level buffer. At the lowest level, data is transferred between the CPU and the highest level buffer in which it is found. If the data is not present in either buffer, the data is transferred directly between the CPU and main memory. A choice among three different load/store instruction pairs, made by a programmer and/or compiler, allows data likely to be used several times to be stored in buffer 15, data likely to be used only a few times to be stored in buffer 16, and data likely to be used only once to be accessed directly from main memory 13, thereby optimising cache use. Cache controller 20 may also implement different preload instructions, one for each level of the cache; and register-clear instructions.

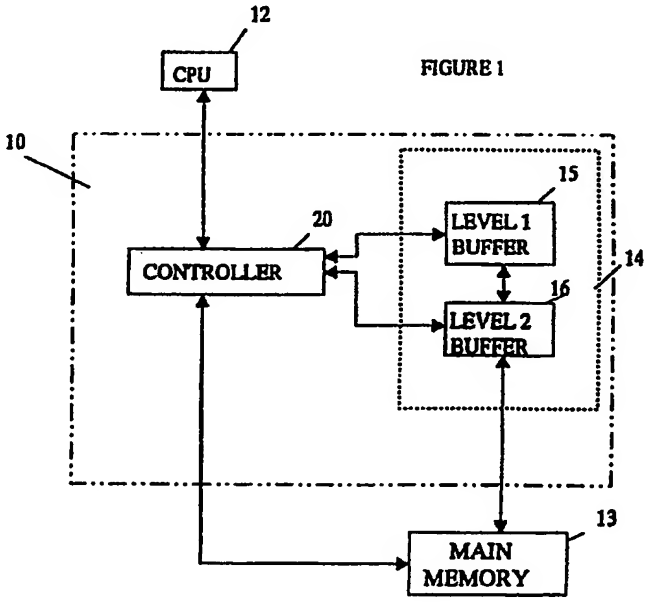

FIGURE 1

GB 2 271 653 A

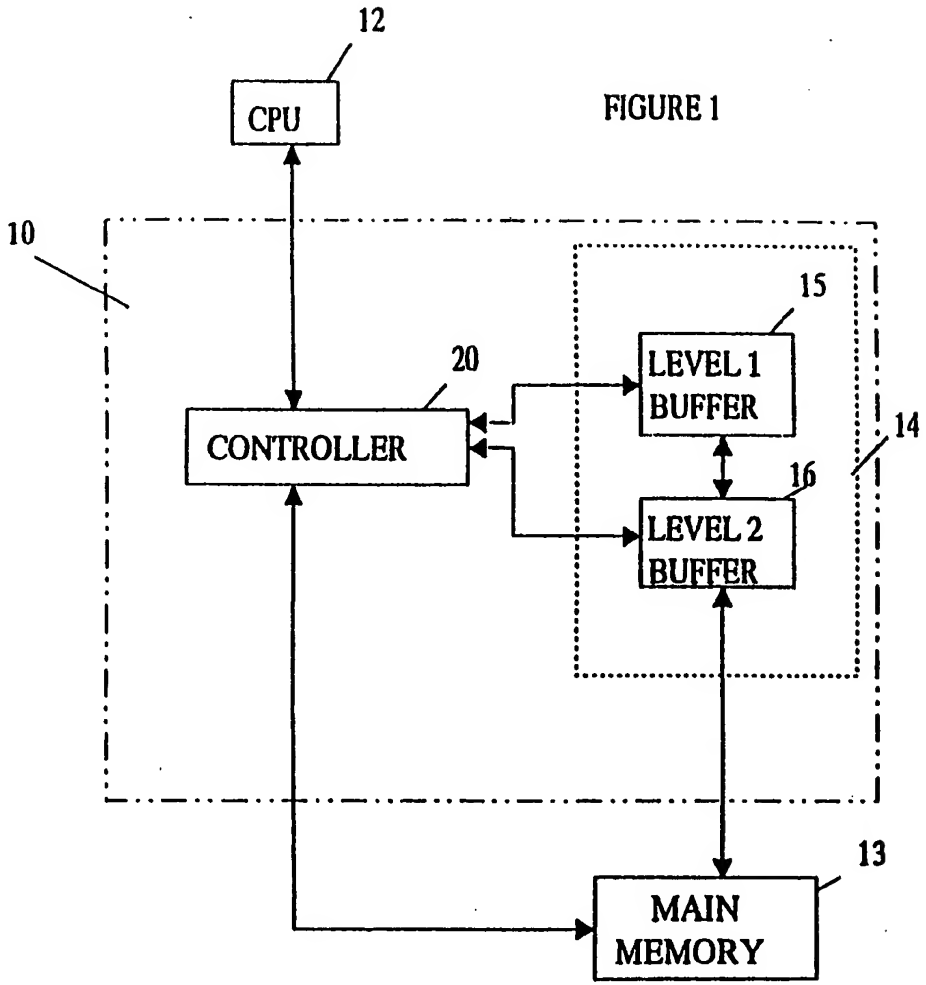

FIGURE 1

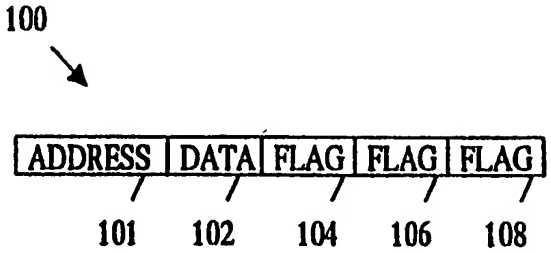

FIGURE 2

MULTI-LEVEL CACHE SYSTEM

The present invention relates to computer memory systems, and more particularly, to an improved cache memory system.

Conventional computer systems utilize memory systems that provide data to the central processing unit (CPU) in response to load instructions and store data into the memory systems in response to store instructions. The cost per computation for the CPU has decreased much faster than the cost per byte of memory. In addition, as computational tasks have become more complex, the size of the main computer memory has dramatically increased. As a result, providing a main memory that operates at the same speed as the CPU has become economically impractical.

To avoid the high cost of providing a main memory that operates at CPU computational speeds, many systems utilize cache memories. A cache is a high speed buffer used to store the most recently used data. When load instructions are issued to the cache, the cache checks it contents. If the data is already present in the cache, the cache returns the data to the CPU. If the data is not present, the cache must load the data from the main memory. Since the main memory is much slower than the cache, this results in a significant delay in the program execution. Each time the cache loads data from the main memory, some of the data stored in the cache must be eliminated to make room for the new data.

Similarly, store instructions are also issued to the cache. If the data for the address specified in the store instruction is already in the cache, the cache updates the data to reflect the values specified in the store instruction. If the data is not present, the cache makes an entry for the address specified in the store instruction and notes the data to be stored at that address. In the case of a "write-through" cache, the data is also sent immediately to the main memory so that the main memory always has a correct copy of the data. In non-write-through cache systems, the data entry in the cache is marked to indicate that it differs from the

value stored at the address in question in the main memory. When a marked data entry is replaced during a subsequent operation, the entry is written to main memory prior to being replaced.

To be effective, the data in the cache must be utilized, on average, a number of times before it is displaced from the cache by new data entering from the main memory in response to load instructions that cannot be satisfied by the data already in the cache. Each time data is acquired from the main memory, the CPU must wait. If the data is used several times while it is in the cache, this delay is amortized over several load instructions; hence, the average delay per load instruction is substantially reduced. No such reduction occurs if the data is used only once.

Cache systems are less effective in large, scientific applications, because there is more uniform access over a much larger set of data in these applications. It is usual, for example, to read sequentially every element in one or more large arrays of data, each of which far exceed the size of the cache. In such cases, data brought into a higher speed cache memory would be accessed only once. As noted above, system performance enhancement is only achieved if the data placed in the cache is used a number of times.

In fact, utilizing the cache for transferring data that is to be used only once actually degrades system performance. As noted above, each time a new data word is moved into the cache from main memory, a data word stored in the cache must be eliminated. Some of the data words that are eliminated would have been used again had those words not been eliminated in response to the load instructions for data words that were to be used only once. When the eliminated data words are again requested, the CPU is delayed while the words are read into the cache. Hence, passing data words that are to be used only once during their residence time in the cache degrades cache performance. This degradation can be reduced by increasing the size of the cache; however, this solution substantially increases the cost of the cache.

For this reason, some machines have been designed to use the cache only for the part of the data which is frequently reused, and to access large arrays independently of the cache. This is common in vector machines, where the vectors

are accessed from highly interleaved memories without passing through a cache. This approach prevents contamination of the cache with data that will only be used once. Unfortunately, this approach does not provide a solution to the problem of data that will be used a few times. Such data would benefit from a cache access scheme, provided it did not displace data from the cache that is likely to be requested a large number of times.

As noted above, cache efficiency increases with the size of the cache. However, to provide the fastest response (i.e., shortest latency), the cache must reside on the processor chip. This limitation sets the maximum size of the cache. In some prior art systems, this conflict of objectives is resolved by utilizing a multi-level hierarchical cache system. A small, level 1 cache is placed directly on the processor chip. This cache is complemented by a large, level 2 cache which is located external to the processor chip.

When data is accessed via prior art multi-level hierarchical caches, the data flows through the level 1 cache. For a read access to memory, if the request can be satisfied by data contained in the level 1 cache, the data is read from the level 1 cache. If the data is not present in the level 1 cache, it is sought from the level 2 cache. If the data can be found in the level 2 cache, it is first moved into the level 1 cache, and then read from the level 1 cache. If the data also cannot be located in the level 2 cache, it is first read from main memory into the level 2 cache, then read from the level 2 cache into level 1 cache, and then read by the processor.

For a write access to memory, if the level 1 cache contains the data presently located at the location to which new data is to be written, the data is updated in the level 1 cache. If the old data is not presently contained in the level 1 cache, room is made in level 1 cache and the old data is sought in the level 2 cache. If the old data is found in the level 2 cache, it is read into the level 1 cache and updated with the newly written data. If the old data also is not contained in the level 2 cache, room is also made in the level 2 cache, the old data is first read into the level 2 cache, it is then read into the level 1 cache, and it is then updated in the level 1 cache.

Since the data flows through the level 1 cache, these schemes suffer from the same contamination problems as conventional caches. That is, data that is used

only a few times displaces data that would have been used many more times. When the replaced data is again requested, the system must be stalled while the data is retrieved from the lower cache level or main memory.

Broadly, it is the object of the present invention to provide an improved cache system.

It is a further object of the present invention to provide a cache system which avoids the problems associated with contamination of the cache with data that is not going to be used a large number of times before it is replaced.

It is a still further object of the present invention to provide a cache system which provides improved access for data that will be used only a few times before being displaced.

These and other objects of the present invention will become apparent to those skilled in the art from the following detailed description of the invention and the accompanying drawings.

The present invention comprises a multi-level cache sub-system for providing buffered memory access between a CPU and a main memory. The invention includes a first buffer for storing copies of data words normally stored in the main memory. This buffer includes a register file for storing information specifying an address in main memory associated with each stored copy. The invention includes a controller receiving first and second store instructions and first and second load instructions from the CPU and for providing data to the CPU in response to the first and second load instructions. Each store and load instruction includes information specifying an address in main memory, the controller is operatively connected to the first buffer and the main memory. The control means also includes first load means, responsive to the first load instruction, for transferring the data word whose address is specified in the first load instruction to the CPU and for causing a copy of the data word to be transferred to the first buffer, and a first store means, responsive to the first store instruction, for causing data included in the store instruction together with the

address information included in the first store instruction to be stored in the first buffer. In addition, the controller includes a second load means, responsive to the second load instruction for transferring the data word whose address is specified in the second load instruction to the CPU from the main memory without causing a copy of the data word to be placed in the first buffer means and a second store means, responsive to the second store instruction, for causing data included in the store instruction to be stored in the main memory without causing a copy of the data word to be placed in the first buffer means.

Other embodiments of the present invention implement additional instructions which further permit the compiler and/or programmer to control the contents of the various buffers.

An exemplary embodiment of the invention will now be described, with reference to the following drawings, in which:

Figure 1 is a block diagram of a cache sub-system according to the present invention.

Figure 2 is a block diagram of a register for storing data words in a cache sub-system according to the present invention.

The present invention may be viewed as a "non-hierarchical" two-level cache subsystem in which control of the level at which data is accessed and stored is controlled by software. A block diagram of a cache sub-system 10 according to the present invention is shown in Figure 1 at 10. Cache sub-system 10 mediates memory accesses between CPU 12 and main memory 13. Cache sub-system 10 includes a controller 20 which interprets load and store instructions from CPU 12 and a buffer 14. Buffer 14 includes a small high-speed level 1 buffer 15 which is preferably located on the same chip as CPU 12. Buffer 14 also includes a somewhat slower but significantly larger level 2 buffer 16. Each of these buffers includes space for storing a plurality of data words and information specifying the address in main memory 12 corresponding to each data word and the length of time since the data word in question was last accessed. In one mode of operation,

buffers 15 and 16 operate in a manner analogous to that described above with reference to prior art hierarchical cache sub-systems.

In general, data that is used the most often is preferably stored in buffer 15, thereby providing CPU 12 with the maximum benefit cache accesses. Data which is likely to be used a few times is preferably stored in buffer 16. As will be explained in more detail below, this provides a means for reducing the effective latency time of computer access to this data while preventing the flow of this data from interfering with the efficient operation of buffer 15. Data that is likely to be used only once is accessed directly from main memory 13.

Cache sub-system 10 differs from prior art devices in that it may access data via three different load/store instruction pairs. The different pairs of instructions allow the system to optimize its performance in terms of the expected reuse of the data being transferred between CPU 12 and main memory 13. Each pair of instructions is utilized for a different level of reuse. The choice of instruction pairs is made by the programmer and/or the compiler. Since the programmer knows the likelihood of data reuse, the programmer can direct the type of memory access utilized. The present invention may also be utilized in conjunction with operating systems which optimize the code by running the code with test data and observing the order in which instructions and data are utilized.

The first pair of load and store instructions accesses data as described above with reference to prior art hierarchical two-level cache subsystems. When controller 20 detects a load instruction of the first kind, controller 20 examines the contents of buffer 15 to determine if the data specified in the load instruction is currently stored in buffer 15. If the data is in buffer 15, the data is delivered to CPU 12 from buffer 15. If the data is not present in buffer 15, controller 20 examines the contents of buffer 16 to determine if the data is stored therein. If the data is found in buffer 16, the data is first copied to buffer 15 and then delivered from buffer 15 to CPU 12. If the data is not found in buffer 16, controller 20 causes copies of the data currently at the address in question in main memory 13 to be placed in both buffers 15 and 16 and then transferred from buffer 15 to CPU 12.

When a store instruction of the first kind is received by controller 20, the data specified in the store instruction is copied into buffer 15. To make room for

the data, the oldest data in buffer 15 is eliminated. Here, age is measured in terms of the number of instruction cycles that have occurred since the data was last referenced. If buffer 15 is not operating as a "write-through" cache, a copy of the eliminated data is made in buffer 16. Any data that must be eliminated from buffer 16 to make room for data received in a store instruction is likewise copied to main memory 13 if necessary. If buffer 15 is operating as a "write-through" cache, this step is not needed since the copy will have already been made the last time the data was accessed. This pair of instructions is preferred for memory accesses involving data that is expected to be used many times before being displaced from the level 1 buffer 15.

The second pair of the load and store instructions never move data higher in the cache hierarchy higher than the level 2 cache buffer shown at 16. When controller 20 detects a load instruction of the second kind, if first examines the contents of buffer 15 to determine if a copy of the requested data word is in buffer 15. If controller 15 finds a copy of the data word in buffer 15, the data word is delivered to CPU 12 from buffer 15. If a copy of the data word is not found in buffer 15, controller 20 examines the contents of buffer 16. If the data word is found in buffer 16, controller 20 causes the data to be delivered to CPU 12 from buffer 16. If the data word is not found in buffer 16, controller 20 causes a copy of the data word to be transferred to buffer 16 from main memory 13. The data word is then delivered to CPU 12 from buffer 16.

When controller 20 detects a store instruction of the second kind, the data word contained therein is copied into buffer 15 if data for the address in question is already present in buffer 15. If buffer 15 is a "write-through" cache, a copy of the data will also be copied to buffer 16. If data for the address is not present in buffer 15, the data word is copied into buffer 16. To make room for the data, the oldest data in buffer 16 may be eliminated. If buffer 16 is not operating as a "write-through" cache, a copy of the eliminated data is made in main memory 13. If buffer 16 is operating as a "write-through" cache, this step is not needed since the copy will have already been made the last time the data was accessed.

This second pair of instructions is preferred for memory accesses involving data that is likely to be reused a small number of times before being displaced from buffer 16. Provided the residency time in buffer 16 is long enough to provide more

than one access to the data in question, the effective latency for the memory access will be reduced. Since this data is not transferred into buffer 15, the problems associated with replacing data in buffer 15 with data that is likely to be used far less often is overcome. The residency time in buffer 16 is related to the size of buffer 16. However, since buffer 16 must only be significantly faster than main memory 13 to provide a significant improvement, buffer 16 can be made much larger than buffer 15 and still provide an economically realistic system.

The third pair instructions never move data higher in the hierarchy above main memory 13. When controller 20 detects a load instruction of the third kind, if first examines buffers 15 and 16 for the data specified in the load instruction. If the data is found in one of these buffers, then the data is transferred from the fastest response time buffer containing the data to CPU 12. If the data is not present in either of the buffers, controller 20 causes the data to be transferred from main memory 13.

When controller 20 detects a store instruction of the third kind, controller 20 first checks buffers 15 and 16 to determine if the data for the address in the store instruction is currently being stored in either buffer. If the data for the address is found in either buffer, the data records in the fastest response time buffer in which a data entry for the address in question exists is updated. If the buffer in question is a "write-through" cache, a copy of the data will also be used to update the slower buffers below the buffer in question in the cache hierarchy. If controller 20 does not find an entry for the data word in question in either buffer, controller 20 causes the data specified in the store instruction to be written directly to main memory 13. This pair of instructions is utilized for data that is not likely to be reused during the time it would reside in buffer 16 if said data were transferred to buffer 16.

In addition to preventing the removal of frequently accessed data from the level 1 and level 2 caches, the present invention provides another important attribute. Compiler technology for scheduling machine instructions can do an excellent job of dealing with long latencies from memory if the compiler knows the latency time. For example, the compiler can send a preload instruction to the cache system at a point sufficiently in advance of the expected load instruction to allow the memory system to move the data into the appropriate level of the cache sub-

system. However, if the preload instruction is sent too early, the data in question may be displaced by data loaded in response to other load and store instructions issued between the time the preload instruction is received and the time the corresponding load instruction arrives. This situation can, in principle, be prevented if the latency time is known. If the latency time is known, the cache system can guarantee that the data is not displaced to early.

What is important is not necessarily the shortest memory latency, but, rather, a predictable latency. A key benefit of the present invention is that it provides more predictable latency times than prior art systems, since it reduces the probability that the data will be removed from the cache buffer in which it was stored before the load instruction accessing the data actually arrives.

In the preferred embodiment of the present invention, controller 20 also implements two preload instructions. one for each level of cache. A preload instruction specifying a memory address and a particular cache causes the data associated with the address in question to be transferred to the cache in question so that the data will be present when a load instruction is received. During the execution of the preload instruction, the cache is free to respond to other load and store instructions. A preload instruction differs from a load instruction in that the CPU will not be stalled until the data is received by the cache. In addition, the preload instruction does not require a register to be specified for the result; hence, a CPU register need not be taken out of service to cause the cache to be loaded with data from a specified address.

The preload instruction must take into account the latency time of the data source to provide the maximum benefit in reducing the effective memory latency time. The preload instruction is preferably issued at least T memory cycles before a load instruction specifying the address in question. Here, T is the latency time of the data source. In a multilevel cache system, the maximum benefit will be achieved if all preloads assume the main memory as the data source.

The manner in which a preload instruction is preferably implemented will be discussed first with reference to the multi-level cache system described with reference to Figure 1. Upon receiving a preload instruction for buffer 15 , controller 20 examines buffer 15 to determine if an entry for the address specified

in the preload instruction is present. If the data is already present, the preload instruction is ignored. If the data is not present, controller 20 examines buffer 16 to determine if the data is there. If it is, the data is moved to buffer 15, if not, a copy of the data is moved first from main memory 13 to buffer 16 and then from buffer 16 to buffer 15. Similarly, upon receiving a preload instruction for buffer 16, controller 20 examines buffer 16 to determine if an entry for the address specified in the preload instruction is present. If the data is already present, the preload instruction is ignored. If the data is not present, controller 20 examines copies the data into buffer 16 from main memory 13.

While a preload is in progress, controller 20 must examine all store instructions issued by the CPU between the receipt of the preload instruction and the delivery of the data into a register in the cache to assure that the most recent data is for the address in question is placed in the cache. Consider the case in which a preload instruction specifying an address A is issued to the cache sub-system, and the data was not present in the cache at any buffer level. During the time needed to move the data from main memory to the cache, a store instruction may be received for address A. If this occurs, the value from the store instruction is to be stored in the cache buffer and the value that is subsequently delivered from main memory is to be ignored.

A second problem introduced by preload instructions relates to the possibility that a second preload instruction for an address A may be received while the cache is still processing the first preload instruction for the same address. If this happens, it is preferred that the cache delays the implementation of the second preload until the first preload instruction is processed. This is situation is expected to occur only rarely; hence, delaying the second preload instruction causes negligible increases in the operating time of the system.

In the preferred embodiment of the present invention, these problems are overcome by adding two additional flag bits to the registers used to store the data values in buffers 15 and 16. In general, buffers 15 and 16 include a register file having one register for each data entry stored in the file. A typical register is shown at 100 in Figure 2. Register 100 includes a field 101 for storing the address in main memory of the data entry stored in field 102. In addition, register 100 includes two flags used in processing preload instructions. The first flag 104

indicates that a preload is in progress. The second flag 106 is used to indicate that a store instruction for the address in question has been received while the preload was in progress, and hence, the value returned from main memory or another buffer level is to be ignored.

Consider the case in which a preload instruction is received and controller 20 determines that the register file in the buffer specified in the instruction does not contain an entry for the address specified in the preload instruction. Controller 20 then assigns a register in the register file of the buffer specified in the preload instruction to the preload instruction. The register flags 104 and 106 are set to their initial values and an instruction fetching the data from the buffer level below the one specified in the preload instruction, or main memory, is issued. Each time a store instruction is detected by controller 20, the address in the store instruction is compared against all of the addresses in registers in the register files. If the address matches, the value specified in the store instruction is loaded into field 102 of the register in question and the second flag is reset to indicate that any value subsequently received from main memory or another buffer is to be ignored. This store instruction processing is in addition to that described above.

The above discussion assumes that the word specified in the store instruction exactly matches the word length of the data entries stored in the buffer. This may not always be the case. For example, in some systems, the cache may store words that are several bytes long; while a store instruction may involve a single byte. In such systems, a separate flag such as flag 106 may be provided for each byte in data field 104. Each such flag would indicate that the corresponding byte had been supplied by a store instruction, and hence, the value for that byte returned in response to the preload instruction is to be ignored.

There are a number of replacement strategies that may be utilized to determine which register in the register file is assigned to the address specified in the preload instruction. Any replacement strategy utilized with a conventional cache may be utilized in a cache system according to the present invention. For example, each register in a register file may include a field for storing a count of the number of cycles that have elapsed since the data entry stored therein was referenced in a store or load instruction. The register having the greatest count would then be assigned to the address in question. This corresponds to replacing

the least used data entry. A second example of a replacement strategy would be to pick a register for which a preload was not in progress at random for replacement.

In addition to the conventional replacement strategies discussed above, the preferred embodiment of the present invention implements a special class of instructions that allow the compiler and/or the programmer to specify which registers may be cleared. This class of instructions will be referred to as clear instructions. There is one such instruction for each buffer level. A clear instruction provides a means for removing data from the cache buffers as soon as the data has been used for the last time by the program. Each instruction specifies an address. If the buffer in question includes an entry for the address in question, then controller 20 marks the register currently being used for this data entry as being free to be overwritten. The register in question can then be used by a preload instruction or a store instruction. That is, a cleared register will be used before replacing the contents of a non-cleared register. To implement these instructions, a third flag 108 is included in each register 100. When controller 20 receives a clear command for an address A and specified buffer, controller 20 examines the contents of the specified buffer for an entry with A in field 101. If such an entry is found, flag 108 is set to a value indicating that the register may be used by a subsequent load, preload, or store operation. If no such register is found, the instruction is ignored. If the cache sub-system is not a write-through cache, controller 20 must also copy the data from the cleared register to the main memory and/or lower level cache buffer when the register is freed.

In principle, a clear instruction can be used with a one level cache to provide a system in which the pollution of the cache by data that is no longer needed in the cache can be avoided. In such a system, the cache buffer must utilize the replacement strategy discussed above, i.e., entries can be marked for replacement. Once a data entry is no longer needed in the cache, a clear instruction directed to the entry in question is issued. If it is known that data will only be used once, each load instruction for such data would be followed by a clear instruction.

While preload instructions allow the compiler and/or programmer to reduce the effects of memory latency time, these instructions increase the number of instructions needed to load a value to the CPU, since each load instruction must now be preceded by a preload instruction. The preferred embodiment of the

present invention avoids this problem by implementing a second class of preload instructions that cause the cache subsystem to effectively generate its own load instruction after a predetermined time delay. This type of preload instruction specifies a buffer level into which the data is to be loaded, the address in main memory of the data entry, the CPU register into which the data is to be loaded, and an instruction cycle count. Controller 20 treats these instructions in the same manner as that described above for preload instructions, except that after the specified number of instructions cycles, controller 20 executes a load instruction for the address in question to the CPU register in question. The preferred embodiment of the present invention implements a second such preload instruction for each buffer level in the cache subsystem.

Similarly, the clear instructions allow the compiler and/or programmer to clear data from the cache that is not likely to be used again prior to being displaced by a subsequent instruction. Such actions are often coupled with either a store or load instruction generated by the last use of the data entry. Hence, instructions which consist of a combination of a store or load instruction followed by a clear instruction are particularly useful, since they avoid issuing two separate instructions. The preferred embodiment of the present invention implements such instructions for each buffer level.

The above-described embodiments of the present invention have utilized two levels of cache buffers. However, it will be apparent to those skilled in the art that systems using more levels of cache buffers may be advantageous. If there are N levels of cache buffers above main memory, then a cache system according to the present invention would include (N+1) load instructions and (N+1) store instructions. The cache level farthest from main memory will be denoted as cache level N, and will be referred to as the highest level in the following discussion. The i^{th} load instruction causes a copy of the data associated with address specified therein to be placed in the i^{th} cache buffer and a copy of that data to be loaded into the CPU register specified in the instruction. The N^{th} cache buffer being the smallest and fastest said level. The 0^{th} load instruction would load the CPU register in question directly from main memory.

Similarly, i^{th} store instruction causes a copy of the data associated with address specified therein to be placed in the i^{th} cache buffer, and 0^{th} store

instruction would store the value in question directly to the main memory. In the case of a store instruction, all entries for the address in a buffer level above the one specified in the instruction must either be invalidated or updated. If the cache subsystem is a write-through cache, an entry must also be placed in each buffer below the specified cache buffer and in main-memory.

The above-described embodiments of the present invention have utilized an "inclusive" cache strategy. That is, copies of the data entries stored in the highest buffer level are also always included in the lower cache buffer levels. For example, when a value is loaded into the i^{th} level buffer, it is also loaded into levels (i-1)...0. Since the size of the cache buffers typically increases by an order of magnitude with each level, the extra copies do not markedly reduce the memory space. The advantage of such an inclusive strategy lies in the ability of controller 20 to determine if an entry is above a specified level by examining the contents of the level in question. If a data entry does not exist at the i^{th} level for address A, then there is no entry at any level less than i. While the preferred embodiment of the present invention utilizes an inclusive cache strategy, it will be apparent to those skilled in the art that non-inclusive strategies may also be utilized without departing from the teachings of the present invention.

While the above-described embodiments of the present invention have assumed at least two levels of buffers in the cache subsystem, it will be apparent to those skilled in the art that a cache subsystem with one level of buffer also provides significant advantages over the prior art. In particular, the inclusion of two levels of store and load instructions, i.e., N=2, allows the compiler and/or programmer to bypass the cache for loads or stores that would otherwise pollute the cache with data that is not likely to be reused during its residency in the cache.

CLAIMS

1. A cache sub-system for providing buffered memory access between a CPU and a main memory, said cache sub-system comprising: buffer means for storing copies of data words normally stored in said main memory, said buffer means comprising means for storing information specifying an address in said main memory associated with each said stored copy; control means for receiving store instructions and load instructions from said CPU and for providing data to said CPU in response to said load instructions, said control means further comprising: load means, responsive to said load instruction, for transferring the data word whose address is specified in said load instruction to said CPU and for causing a copy of said data word to be transferred to said buffer means; and store means, responsive to said store instruction, for causing data included in said store instruction together with said address information included in said store instruction to be stored in said first buffer means.

2. A cache sub-system for providing buffered memory access between a CPU and a main memory, said cache sub-system comprising: first buffer means for storing copies of data words normally stored in said main memory, said first buffer means further comprising means for storing information specifying an address in said main memory, said first buffer means further comprising means for storing information specifying an address in said main memory associated with each said stored copy; control means for receiving first and second store instructions and first and second load instructions from said CPU and for providing data to said CPU in response to said first and second load instructions, each said store and load instruction comprising information specifying an address

in said main memory, said control means being operatively connected to said first buffer means and said main memory, said control means further comprising: first load means, responsive to said first load instruction, for transferring the data word whose address is specified in said fist load instruction to said CPU and for causing a copy of said data word to be transferred to said first buffer means; first store means, responsive to said first store instruction, for causing data included in said store instruction together with said address information included in said first store instruction to be stored in said first buffer means; second load means, responsive to said second load instruction for transferring the data word whose address is specified in said second load instruction to said CPU from said main memory without causing a copy of said data word to be placed in said first buffer means; and second store means, responsive to said second store instruction, for causing data included in said store instruction to be stored in said main memory without causing a copy of said data word to be placed in said first buffer means.

3. The cache sub-system of claim 2, wherein said control means further comprises means, responsive to the receipt of a preload instruction including an address, for causing a copy of the most recent data associated with said address in said main memory to be stored in said first buffer means while allowing load and store instructions to be processed while said preload instruction is in progress.

4. The cache sub-system of claims 2 or 3, wherein said control means, responsive to the receipt of a clear instruction including an address, for causing any copy of data associated with said address to be removed from

said first buffer means.

5. The cache sub-system of any of claims 2 to 4, further comprising: second buffer means for storing copies of data words normally stored in said main memory, said second buffer means further comprising means for storing information specifying the address of each said stored data word in said main memory, wherein said control means is operatively connected to said second buffer means and said control means further comprises: means for receiving a third store instruction and a third load instruction form said CPU and for providing data to said CPU in response to said third load instructions and receiving data from said CPU in response to said third store instruction; third store means, responsive to said third store instruction, for causing data included in said store instruction together with said address information included in said third store instruction to be stored in said second buffer means; and third load means, responsive to said third load instruction for transferring the data whose address is specified in said third load instruction to said CPU and for causing a copy of said data word to be stored in said second buffer means.

6. The cache sub-system of claim 5, wherein said first store also causes a copy of said data included in said first store instruction together with said address information included in said first store instruction to be stored in said second buffer means; and said first load means also causes a copy of said data included in said first load instruction together with said address information included in said first load instruction to be stored in said second buffer means.

7. A cache sub-system for providing buffered memory access between a CPU and a main memory, said cache sub-system comprising: buffer means for storing copies of data words normally stored in said main memory, said first buffer means further comprising means for storing information specifying an address in said main memory associated with each said stored copy; control means for receiving first store instructions and first load instruction from said CPU and for providing data to said CPU in response to said load instructions, each said store and load instruction comprising information specifying an address in said main memory, said control means being operatively connected to said buffer means and said main memory,said control means further comprising: load means, responsive to said load instruction, for transferring the data word whose address is specified in said load instruction to said CPU and for causing a copy of said data word to be transferred to said buffer means; store means, responsive to said store instruction, for causing data included in said store instruction together with said address information included in said store instruction to be store in said buffer mean; and clear means, responsive to a clear instruction including an address, for causing any copy of a data word in said buffer means corresponding to said address to be cleared from said buffer means.

8. The cache sub-system of claim 7, wherein said control means further comprises means, responsive to a second load instruction for transferring the data word whose address is specified in said second load instruction to said CPU and for causing any copy of said data word in said buffer means to be cleared form said buffer means.

9. The cache sub-system of claims 7 or 8, wherein said control means comprises means, responsive to a second store instruction for causing data included in said store instruction together with said address information included in said store instruction to be stored in said main memory and for causing any copy of said data wore in said buffer means to be cleared form said buffer means.

10. A cache sub-system for providing buffered memory access between a CPU and a main memory, said cache sub-system comprising: first buffer means for storing copies of data words normally stored in said main memory, said first buffer means further comprising means for storing information specifying an address in said main memory associated with each said stored copy; control means for receiving store instructions and load instructions form said CPU and for providing data to said CPU in response to said load instructions, each said store and load instruction comprising information specifying an address in said main memory, said control means being operatively connected to said first buffer means and said main memory, said control means further comprising: load means, responsive to said load instruction, for transferring the data word whose address is specified in said load instruction to said CPU and for causing a copy of said data word to be transferred to said first buffer means; store means, responsive to said store instruction, for causing data included in said store instruction together with said address information included in said store instruction to be stored in said first buffer means; and preload means, responsive to a first preload instruction including an address, for causing a copy of the most recent data associated with said address in said main memory to be stored in said first buffer means while

allowing load and store instructions to be processed while said preload instruction is in progress.

11. The cache sub-system of claim 10, wherein said control means further comprises further preload means, responsive to a second preload instruction including an address, for causing a copy of the most recent data associated with said address in said main memory to be stored in said first buffer means while allowing load and store instructions to be processed while said preload instruction is in progress and for causing a copy of said data to be transferred to said CPU after a predetermined delay.

12. A cache sub-system substantially as herein described with reference to the accompanying drawings.

13. A method of controlling communication between a CPU and a main memory, the method comprising the steps of providing first buffer means for storing copies of data words normally stored in said main memory, storing in the first buffer means information specifying an address in said main memory associated with each stored copy; providing data to said CPU in response to first and second load instructions, each said store and load instruction comprising information specifying an address in said main memory; transferring the data word whose address is specified in said first load instruction to said CPU for causing a copy of said data word to be transferred to said first buffer means; transferring the data word whose address is specified in said second load instruction to said CPU from said main memory without causing a copy of said data word to be placed in said first buffer means; and causing data included in said store instruction to be stored in said main memory without causing a copy of said data word to be placed in

said first buffer means.

14. A method of controlling communication between a CPU and a main memory substantially as herein described.

Patents Act 1977
Examiner's report to the Comptroller under Section 17
(The Search report)

-22-

Application number
GB 9320511.0

Relevant Technical Fields

(i) UK Cl (Ed.L) G4A (AMC)

(ii) Int Cl (Ed.5) G06F 12/08

Search Examiner
B G WESTERN

Date of completion of Search
17 NOVEMBER 1993

Databases (see below)
(i) UK Patent Office collections of GB, EP, WO and US patent specifications.

(ii)

Documents considered relevant following a search in respect of Claims :-
1-6, 12-14

Categories of documents

X: Document indicating lack of novelty or of inventive step.

Y: Document indicating lack of inventive step if combined with one or more other documents of the same category.

A: Document indicating technological background and/or state of the art.

P: Document published on or after the declared priority date but before the filing date of the present application.

E: Patent document published on or after, but with priority date earlier than, the filing date of the present application.

&: Member of the same patent family; corresponding document.

Category	Identity of document and relevant passages		Relevant to claim(s)
X	GB 1449877 A	(SIEMENS) see whole document	1,2,13
X	GB 2037038 A	(HONEYWELL) NB pages 1-11, Figures 1-5	1,2,13
X	EP 0077451 A2	(IBM) see whole document	1,2,13
X	US 4075686 A	(CALLE ET AL) NB columns 17-37	1,2,4,13
X	US 4197580 A	(CHANG ET AL) see whole document	1,2,13

Databases:The UK Patent Office database comprises classified collections of GB, EP, WO and US patent specifications as outlined periodically in the Official Journ (Patents). The on-line databases considered for search are also listed periodically in the Official Journal (Patents).